SYSTEM AND METHOD FOR TRACKING CHANGES TO FILES IN STREAMING APPLICATIONS

BACKGROUND

When streaming a software title a client is led to believe that some or all of the files associated with the streaming software title are locally available. The files actually may or may not reside on the local system. Techniques are used to trick the client into believing that all of the files are available. A description of streaming software is provided with reference to U.S. Pat. No. 6,453,334 filed on June 16, 1998.

A client, believing that a file associated with a streaming title is available, may attempt to open the file. A file may be opened for read-only access, write-only access, or read/write access. In the case of read-only access, the file can be accessed according to streaming protocols. In the case of a write-only access, the file can be created locally. In the case of a read/write access, the file is downloaded, and then modified locally.

It may be desirable to use read/write access to open a file, but not download the entire file.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 depicts a flowchart of an example of a method for dealing with a read/write file in a streaming software context.

FIG. 2 depicts a flowchart of an example of a method for dealing with a read/write file in a streaming software context.

FIG. 3 depicts a flowchart of an example of a method for dealing with modified segments of a file in a streaming software context.

FIG. 4 depicts a flowchart of an example of a method for dealing with a modification request in a streaming software context.

FIG. 5 depicts an example of a system 500 that includes a diff-file on a streaming client.

FIG. 6 depicts a flowchart 600 of an example of a method for virtually representing a file on a client with local modifications.

FIG. 7 depicts a networked system for use in an embodiment.

FIG. 8 depicts a computer system for use in the system of FIG. 7.

DETAILED DESCRIPTION

One technique for streaming software is described in the co-pending patent application no. 10/988,014 filed November 11, 2004, entitled "SYSTEM AND METHOD FOR PREDICTIVE STREAMING".

FIG. 1 depicts a flowchart 100 of an example of a method for dealing with a read/write file in a streaming software context. This method and other methods are depicted as serially arranged modules. However, modules of the methods may be reordered, or arranged for parallel execution as appropriate. The flowchart 100 starts at module 102 wherein a file associated with a streaming software program is opened for modification. Typically, files may be opened read-only, write-only, or read/write. A file that is opened read/write may be referred to as a file opened for modification. Files that may or may not be stored locally when opened may be referred to as "virtual files." Advantageously, read/write and write-only files can be virtual files by using techniques described herein.

When a file is opened write-only, the file is often a new file. For example, in a streaming context, if a client opens a file write-only, the client will typically create the new file locally. When a file is opened read-only, the file, or portions of the file may be streamed as needed. Portions of the file may also be streamed according to predicted need, as is described in the co-pending patent application no. 10/988,014 filed November 11, 2004, entitled "SYSTEM AND METHOD FOR PREDICTIVE STREAMING." When a file is opened read/write, on the other hand, the file may or may not be written to. For example, a client may have the right to modify (write) a file, but have no need to modify the file.

In the example of FIG. 1, the flowchart 100 continues at decision point 104 where it is determined whether the file is written to. It has been determined that in many cases, files are opened read/write, but then treated as read-only. Advantageously, in a non-limiting embodiment, if it is determined that the file has not been written to (104-N), then the flowchart 100 continues at module 106 where the file is treated as read-only until the file is written to. The flowchart 100 ends if the file is never written to. This can reduce the need to download portions of the file in preparation for writes.

If it is determined that the file is written to (104-Y), then the flowchart 100 continues at module 108, where changes to the file are tracked locally. For example, if, according to a write procedure, a pointer seeks to a location of the file and writes at that location, then the written part will be recorded locally, along with, for example, an indication of the location of the write. In an embodiment, the locally tracked information includes by way of example but not limitation, the

size of the write, the location of the write, and an end-of-file (eof) indicator. The eof indicator can be valuable when a truncating write is performed on a file. Keeping track of the size of the write is optional, since the size of the write is determinable from the size of the written data.

In the example of FIG. 1, the flowchart 100 continues at module 110, wherein the file is closed. The file may be closed at any time during execution of an associated streaming program, or when the streaming program or some other associated program ends. This is not intended to limit the method to programs that close the file, since a system could be designed that allows the file to remain open practically all the time and the techniques described herein would still be applicable. Moreover, use of the file is not necessarily limited at any stage of the flowchart 100, before, after, or between modifications.

It may be noted that no current API exists to delete from the middle of a file. Rather, what appears to be a deletion from the middle of a file is actually a truncating write. This is not intended to limit the invention in any way, since at least some embodiments of the invention would still work if an API were designed to perform a delete from the middle of a file.

FIG. 2 depicts a flowchart 200 of an example of a method for dealing with a read/write file in a streaming software context. The flowchart 200 starts at decision point 202 where it is determined whether a file has been modified. For the purposes of this example, it is assumed that the file has been opened read/write. If the file is not modified (202-N), then the flowchart 100 continues at module 204, where the file is treated as a read-only file. In some cases, a file that is opened read/write is not modified from the time the file is opened until the file is closed (at module 212).

If, on the other hand, the file is modified (202-Y), then the flowchart 200 continues at decision point 206 where it is determined whether the size of the file is less than a writeback threshold. The writeback threshold is a value of arbitrary size that is used to determine whether it is "worth it" to keep track of changes to a file. The threshold may be set to, by way of example but not limitation, 250KB. In some cases it may be desirable to increase or decrease this threshold depending upon such factors as local memory resources and download bandwidth.

If it is determined that the size of the file is less than the writeback threshold (206-Y), then at module 208 the file is downloaded and modified locally. If, on the other hand, it is determined that the size of the file is not less than the writeback threshold (206-N), then at module 210 changes to the file are tracked locally in lieu of downloading and modifying the file. In this way, changes to large files can be tracked locally without using up bandwidth or memory

in an attempt to download the large files. In any case, the file is assumed to eventually be closed at module 212, and the flowchart 100 ends.

FIG. 3 depicts a flowchart 300 of an example of a method for dealing with modified segments of a file in a streaming software context. The flowchart 300 starts at module 302 where changes to segments of a file are tracked locally. The flowchart 300 continues at module 304 where overlapping segments of the file are merged. In a non-limiting embodiment, if the tracked changes result in changes to a segment twice, then the latest change to the segment should control. In order to save space and/or reduce the risk of error, the two changes should be merged into a single change.

The method of FIG. 3 could be incorporated into the method of FIG. 2. For example, for files that exceed a writeback threshold, segments of the file could be tracked locally, and overlapping segments could be merged. Alternatively, all files could be handled as described with reference to FIG. 3. Alternatively, the example of FIG. 3 could be applied to files that have a specified characteristic, such as, by way of example but not limitation, certain media files, token files, files associated with a certain application, etc.

FIG. 4 depicts a flowchart 400 of a method according to an embodiment. The flowchart 400 starts at module 402 where a file modify request is received. This request may follow an open read/write of the file.

In an embodiment, the flowchart 400 continues at decision point 404 where it is determined whether the file is associated with an application in which an append is typical. By way of example but not limitation, many games have data files that are modified over time. Typically, the data files include saved state. As the game progresses, additional saved state is appended to the data file. Thus, a game file may be associated with an application in which an append is typical. Other files, such as Word documents, may not typically be appended to (modifications may occur with reasonable frequency anywhere in the file). Accordingly, in some cases, it may be desirable to "turn off" the diff-file procedures.

In the example of FIG. 4, if it is determined that the file is not associated with an application in which append is typical (404-N), then at module 406 the file is downloaded, at module 408 the file is modified according to the modify file request, and the flowchart 400 ends.

If, on the other hand, it is determined that the file is associated with an application in which append is typical (404-Y), then at module 410 the file is modified according to a diff-file technique and the modify file request, then the flowchart 400 ends. The diff-file technique

involves tracking changes to the file locally without downloading the file in its entirety, examples of which are provided with reference to FIGS. 1-3.

FIG. 5 depicts an example of a system 500 that includes a diff-file on a streaming client. The system 500 includes a server 502 with a file 504 stored thereon, a network 506, and a client 510 with a diff-file 512 and a diff-file integration engine 514 stored thereon. The server 502 and the client 510 will include other components, such as a processor, as is described later with reference to FIG. 8. In the example of FIG. 5, the server 502 may be referred to as a streaming server and the client 510 may be referred to as a streaming client in the context of a streaming software system. The diff-file 512 includes locally tracked changes to the file 504. The diff-file integration engine 514 combines the file 504 and the diff-file 512 in such a way that the file 504, modified as indicated in the diff-file 512, appears to reside locally on the client 502.

It should be noted that the file 504 may be referred to as a virtual file on the client 504. That means the client 502 includes one or more of a virtual environment in which to execute the file 504, registry information that has been spoofed to trick the client 502 into believing the file 504 is stored locally, or some other mechanism that makes the client 502 pretend that the file 504 is stored locally when it is not. Moreover, the file 504 could eventually be stored locally. For example, the file 504 might initially not be stored locally, but over time the client 502 might download some or all of the file 504 in, for example, streamed blocks. As used herein, the file 504 is referred to as a virtual file if it is capable of being a virtual file, even if it is eventually stored local to the client.

FIG. 6 depicts a flowchart 600 of an example of a method for virtually representing a file on a client with local modifications. The flowchart 600 starts at module 602 wherein a file is virtually represented on a streaming client. The virtual representation of a file on a streaming client may involve modifying a system registry, or presenting the file in a virtual environment. A file that is represented virtually need not actually exist on the streaming client. This is typical of streaming software applications, and allows the streamed applications to be executed locally without downloading the application in its entirety, and without installing the application.

In the example of FIG. 6, the flowchart 600 continues at module 604 wherein updates to the file are written to a diff-file on the streaming client. In this way, even though the file has not been downloaded to the streaming client, and the streamed file remains unchanged at a server, the file can be "virtually modified" by tracking changes that would have been made to the file if the file were stored locally.

The following description of FIGS. 7 and 8 is intended to provide an overview of computer hardware and other operating components suitable for performing the methods of the invention described herein, but is not intended to limit the applicable environments. Similarly, the computer hardware and other operating components may be suitable as part of the apparatuses of the invention described herein. The invention can be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The invention can also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network.

FIG. 7 depicts a networked system 700 that includes several computer systems coupled together through a network 702, such as the Internet. The term "Internet" as used herein refers to a network of networks which uses certain protocols, such as the TCP/IP protocol, and possibly other protocols such as the hypertext transfer protocol (HTTP) for hypertext markup language (HTML) documents that make up the World Wide Web (the web). The physical connections of the Internet and the protocols and communication procedures of the Internet are well known to those of skill in the art.

The web server 704 is typically at least one computer system which operates as a server computer system and is configured to operate with the protocols of the world wide web and is coupled to the Internet. The web server system 704 can be a conventional server computer system. Optionally, the web server 704 can be part of an ISP which provides access to the Internet for client systems. The web server 704 is shown coupled to the server computer system 706 which itself is coupled to web content 708, which can be considered a form of a media database. While two computer systems 704 and 706 are shown in FIG. 7, the web server system 704 and the server computer system 706 can be one computer system having different software components providing the web server functionality and the server functionality provided by the server computer system 706, which will be described further below.

Access to the network 702 is typically provided by Internet service providers (ISPs), such as the ISPs 710 and 716. Users on client systems, such as client computer systems 712, 718, 722, and 726 obtain access to the Internet through the ISPs 710 and 716. Access to the Internet allows users of the client computer systems to exchange information, receive and send e-mails, and view documents, such as documents which have been prepared in the HTML format. These documents are often provided by web servers, such as web server 704, which are referred to as

being "on" the Internet. Often these web servers are provided by the ISPs, such as ISP 710, although a computer system can be set up and connected to the Internet without that system also being an ISP.

Client computer systems 712, 718, 722, and 726 can each, with the appropriate web browsing software, view HTML pages provided by the web server 704. The ISP 710 provides Internet connectivity to the client computer system 712 through the modem interface 714, which can be considered part of the client computer system 712. The client computer system can be a personal computer system, a network computer, a web TV system, or other computer system. While FIG. 7 shows the modem interface 714 generically as a "modem," the interface can be an analog modem, isdn modem, cable modem, satellite transmission interface (e.g. "direct PC"), or other interface for coupling a computer system to other computer systems.

Similar to the ISP 714, the ISP 716 provides Internet connectivity for client systems 718, 722, and 726, although as shown in FIG. 7, the connections are not the same for these three computer systems. Client computer system 718 is coupled through a modem interface 720 while client computer systems 722 and 726 are part of a LAN 730.

Client computer systems 722 and 726 are coupled to the LAN 730 through network interfaces 724 and 728, which can be Ethernet network or other network interfaces. The LAN 730 is also coupled to a gateway computer system 732 which can provide firewall and other Internet-related services for the local area network. This gateway computer system 732 is coupled to the ISP 716 to provide Internet connectivity to the client computer systems 722 and 726. The gateway computer system 732 can be a conventional server computer system.

Alternatively, a server computer system 734 can be directly coupled to the LAN 730 through a network interface 736 to provide files 738 and other services to the clients 722 and 726, without the need to connect to the Internet through the gateway system 732.

FIG. 8 depicts a computer system 740 for use in the system 700 (FIG. 7). The computer system 740 may be a conventional computer system that can be used as a client computer system or a server computer system or as a web server system. Such a computer system can be used to perform many of the functions of an Internet service provider, such as ISP 710 (FIG. 7).

In the example of FIG. 8, the computer system 740 includes a computer 742, I/O devices 744, and a display device 746. The computer 742 includes a processor 748, a communications interface 750, memory 752, display controller 754, non-volatile storage 756, and I/O controller 758. The computer system 740 may be couple to or include the I/O devices 744 and display device 746.

The computer 742 interfaces to external systems through the communications interface 750, which may include a modem or network interface. It will be appreciated that the communications interface 750 can be considered to be part of the computer system 740 or a part of the computer 742. The communications interface can be an analog modem, isdn modem, cable modem, token ring interface, satellite transmission interface (e.g. "direct PC"), or other interfaces for coupling a computer system to other computer systems.

The processor 748 may be, for example, a conventional microprocessor such as an Intel Pentium microprocessor or Motorola power PC microprocessor. The memory 752 is coupled to the processor 748 by a bus 760. The memory 752 can be dynamic random access memory (DRAM) and can also include static ram (SRAM). The bus 760 couples the processor 748 to the memory 752, also to the non-volatile storage 756, to the display controller 754, and to the I/O controller 758.

The I/O devices 744 can include a keyboard, disk drives, printers, a scanner, and other input and output devices, including a mouse or other pointing device. The display controller 754 may control in the conventional manner a display on the display device 746, which can be, for example, a cathode ray tube (CRT) or liquid crystal display (LCD). The display controller 754 and the I/O controller 758 can be implemented with conventional well known technology.

The non-volatile storage 756 is often a magnetic hard disk, an optical disk, or another form of storage for large amounts of data. Some of this data is often written, by a direct memory access process, into memory 752 during execution of software in the computer 742. One of skill in the art will immediately recognize that the terms "machine-readable medium" or "computer-readable medium" includes any type of storage device that is accessible by the processor 748 and also encompasses a carrier wave that encodes a data signal.

Objects, methods, inline caches, cache states and other object-oriented components may be stored in the non-volatile storage 756, or written into memory 752 during execution of, for example, an object-oriented software program. In this way, the components illustrated in, for example, FIGS. 1-3 and 6 can be instantiated on the computer system 740.

The computer system 740 is one example of many possible computer systems which have different architectures. For example, personal computers based on an Intel microprocessor often have multiple buses, one of which can be an I/O bus for the peripherals and one that directly connects the processor 748 and the memory 752 (often referred to as a memory bus). The buses are connected together through bridge components that perform any necessary translation due to differing bus protocols.

Network computers are another type of computer system that can be used with the present invention. Network computers do not usually include a hard disk or other mass storage, and the executable programs are loaded from a network connection into the memory 752 for execution by the processor 748. A Web TV system, which is known in the art, is also considered to be a computer system according to the present invention, but it may lack some of the features shown in FIG. 8, such as certain input or output devices. A typical computer system will usually include at least a processor, memory, and a bus coupling the memory to the processor.

In addition, the computer system 740 is controlled by operating system software which includes a file management system, such as a disk operating system, which is part of the operating system software. One example of an operating system software with its associated file management system software is the family of operating systems known as Windows® from Microsoft Corporation of Redmond, Washington, and their associated file management systems. Another example of operating system software with its associated file management system software is the Linux operating system and its associated file management system. The file management system is typically stored in the non-volatile storage 756 and causes the processor 748 to execute the various acts required by the operating system to input and output data and to store data in memory, including storing files on the non-volatile storage 756.

Some portions of the detailed description are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates

and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

The present invention, in some embodiments, also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the methods of some embodiments. The required structure for a variety of these systems will appear from the description below. In addition, the present invention is not described with reference to any particular programming language, and various embodiments may thus be implemented using a variety of programming languages.

While this invention has been described in terms of certain embodiments, it will be appreciated by those skilled in the art that certain modifications, permutations and equivalents thereof are within the inventive scope of the present invention. It is therefore intended that the following appended claims include all such modifications, permutations and equivalents as fall within the true spirit and scope of the present invention; the invention is limited only by the claims.

CLAIMS

What is claimed is:

1. A method, comprising:
virtually representing a file on a streaming client;
writing modifications to the virtual file into a diff-file on the streaming client.

2. The method of claim 1, wherein the modifications are made locally, further comprising integrating the virtual file with the diff-file, yielding a locally modified virtual file.

3. The method of claim 1, further comprising tracking segments of the file locally.

4. The method of claim 3, further comprising merging overlapping segments of the file.

5. The method of claim 1, further comprising:
receiving a modify file request;
modifying the file according to a diff-file technique and the modify file request if the file is associated with an application for which an append is typical.

6. The method of claim 1, further comprising:
receiving a modify file request;
if the file is associated with an application in which an append is atypical:
downloading the file;
modifying the file according to the modify file request.

7. The method of claim 1, further comprising, if file size is less than a writeback threshold:
downloading a file associated with the virtual file;
modifying the downloaded file locally.

8. A method comprising:
opening for modification a virtual file associated with a streaming software program;
treating the virtual file as a read-only file until a modification is received;
receiving a modification to the virtual file;
storing the modification in a local file;
integrating the modification into the virtual file when the virtual file is used.

9. The method of claim 8, further comprising tracking segments of the file locally.

10. The method of claim 9, further comprising merging overlapping segments of the file.

11. The method of claim 8, further comprising:
receiving a modify file request;
modifying the virtual file according to a diff-file technique and the modify file request if the virtual file is associated with an application for which an append is typical.

12. The method of claim 8, further comprising:
receiving a modify file request;
if the virtual file is associated with an application in which an append is atypical:
downloading the virtual file;
modifying the downloaded virtual file according to the modify file request.

13. The method of claim 8, further comprising, if file size is less than a writeback threshold:
downloading the virtual file;
modifying the downloaded virtual file locally.

14. A system, comprising:
a virtual file associated with a remotely stored file;
a diff-file, stored locally, associated with the remotely stored file, wherein the diff-file includes local changes to the virtual file;
a diff-file integration engine capable of combining the virtual file with the diff-file to create a locally modified virtual file.

15. The system of claim 14, wherein the diff-file is further used to track segments of the virtual file locally.

16. The system of claim 14, wherein the diff-file is further used to merge overlapping segments of the virtual file.

17. The system of claim 14, further comprising a means for receiving a modify file request, wherein the diff-file integration engine modifies the virtual file according to a diff-file technique and the modify file request, if the virtual file is associated with an application for which an append is typical.

18. The system of claim 14, further comprising:

a means for receiving a modify file request

a means for downloading the virtual file if the virtual file is associated with an application in which an append is atypical;

a means for modifying the file according to the modify file request if the virtual file is associated with an application in which an append is atypical.

19. The system of claim 14, further comprising, if file size is less than a writeback threshold:

a means for downloading a file associated with the virtual file if file size is less than a writeback threshold;

a means for modifying the downloaded file locally if file size is less than a writeback threshold.

20. The system of claim 14, further comprising:

memory having a plurality of modules stored therein;

a processor, coupled to the memory, capable of executing the executable modules,

wherein the memory includes the virtual file, the diff-file, and the diff-file integration engine.

ABSTRACT

A technique for modifying virtual files involves tracking changes locally. A method according to the technique may include virtually representing a file on, for example, a streaming client, and writing modifications to the virtual file into a diff-file on the streaming client. A system according the technique may include a virtual file associated with a remotely stored file, a diff-file, stored locally, associated with the remotely stored file, and a diff-file integration engine. The diff-file may include local changes to the virtual file. The diff-file integration engine may be capable of combining the virtual file with the diff-file to create a locally modified virtual file.

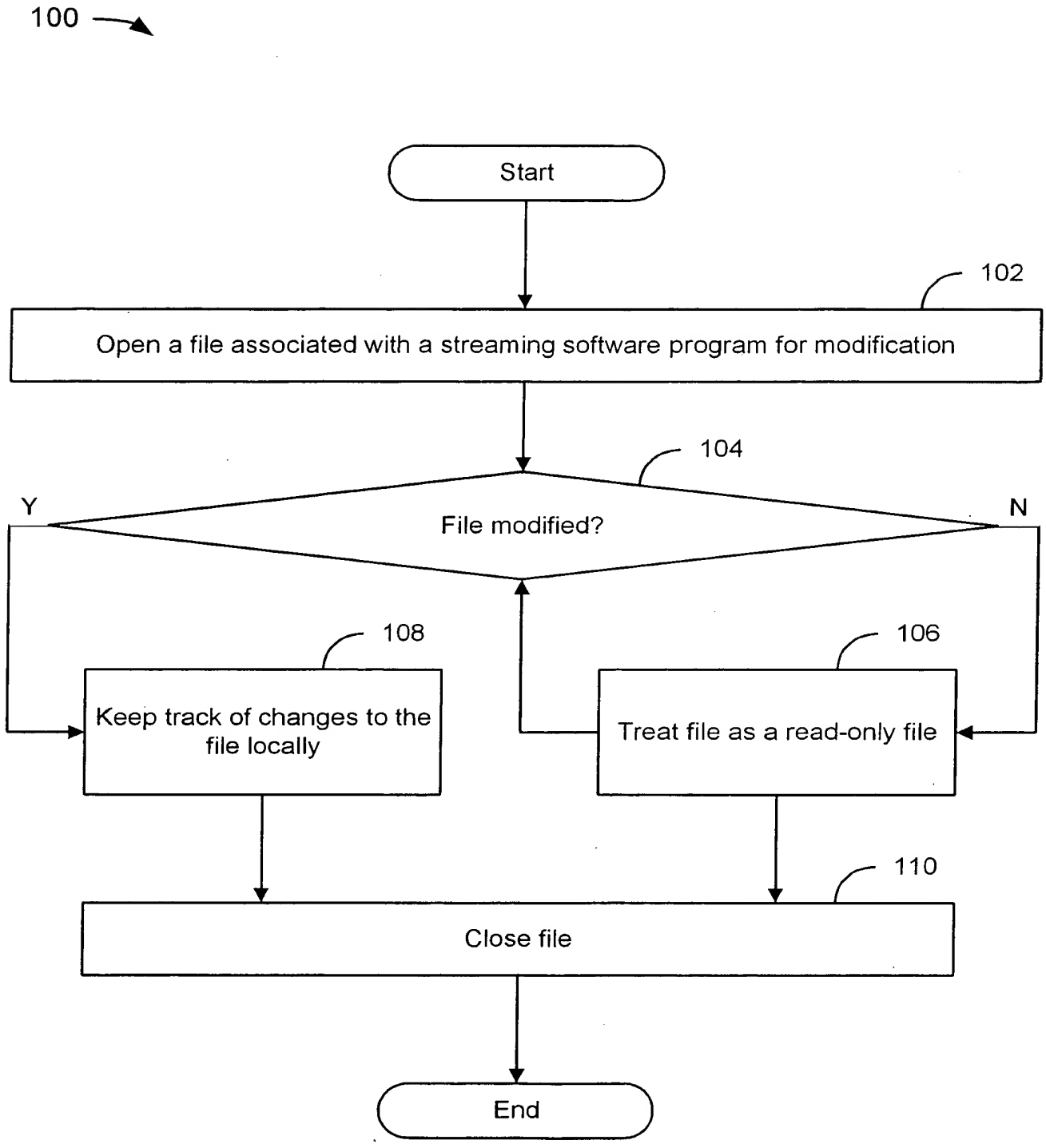
100
Start
102
Open a file associated with a streaming software program for modification
104
File modified?
Y
N
108
Keep track of changes to the file locally
106
Treat file as a read-only file
110
Close file
End

FIG. 1

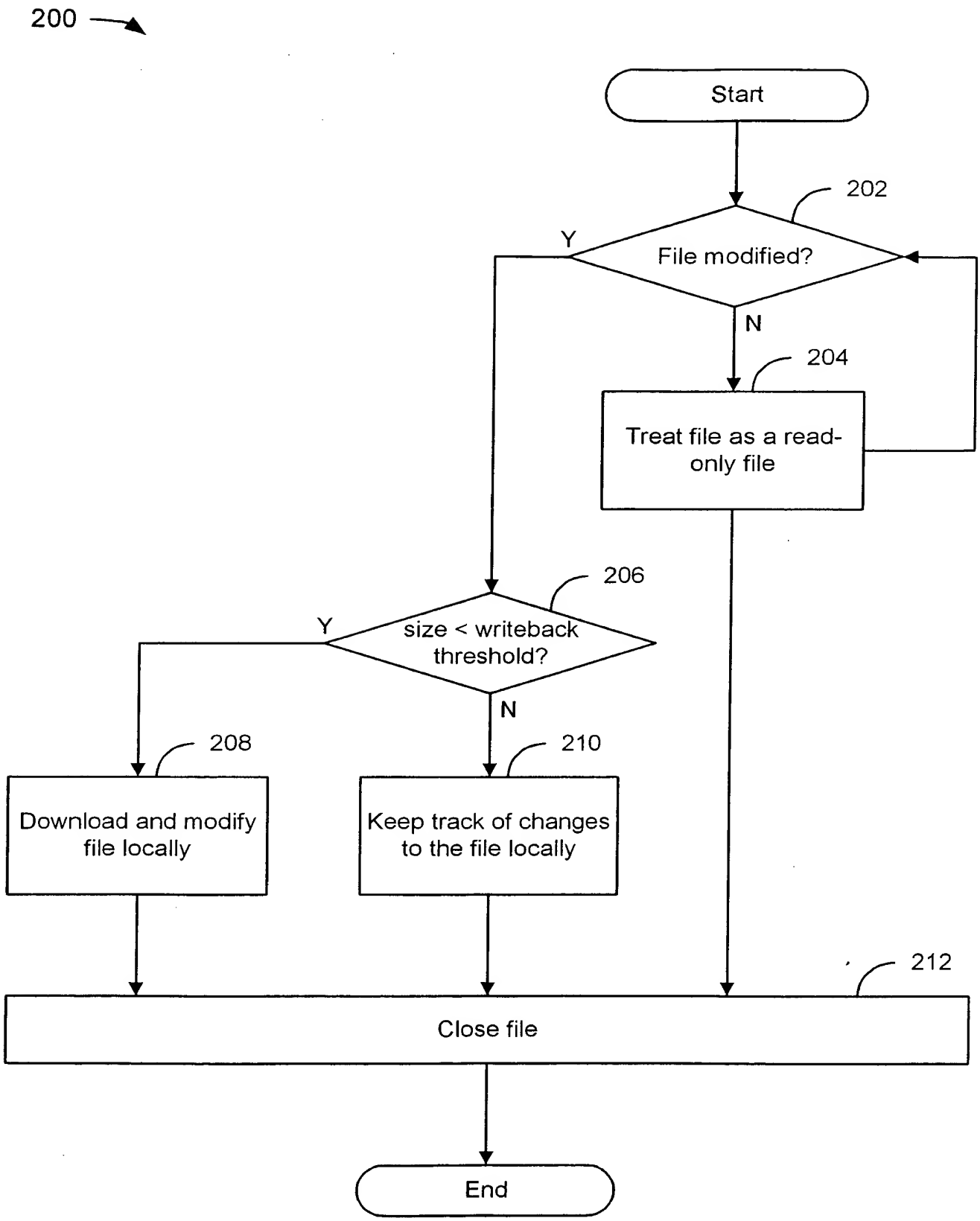
200
Start
202
File modified?
Y
N
204
Treat file as a read-only file
206
size < writeback threshold?
Y
N
208
Download and modify file locally
210
Keep track of changes to the file locally
212
Close file
End

FIG. 2

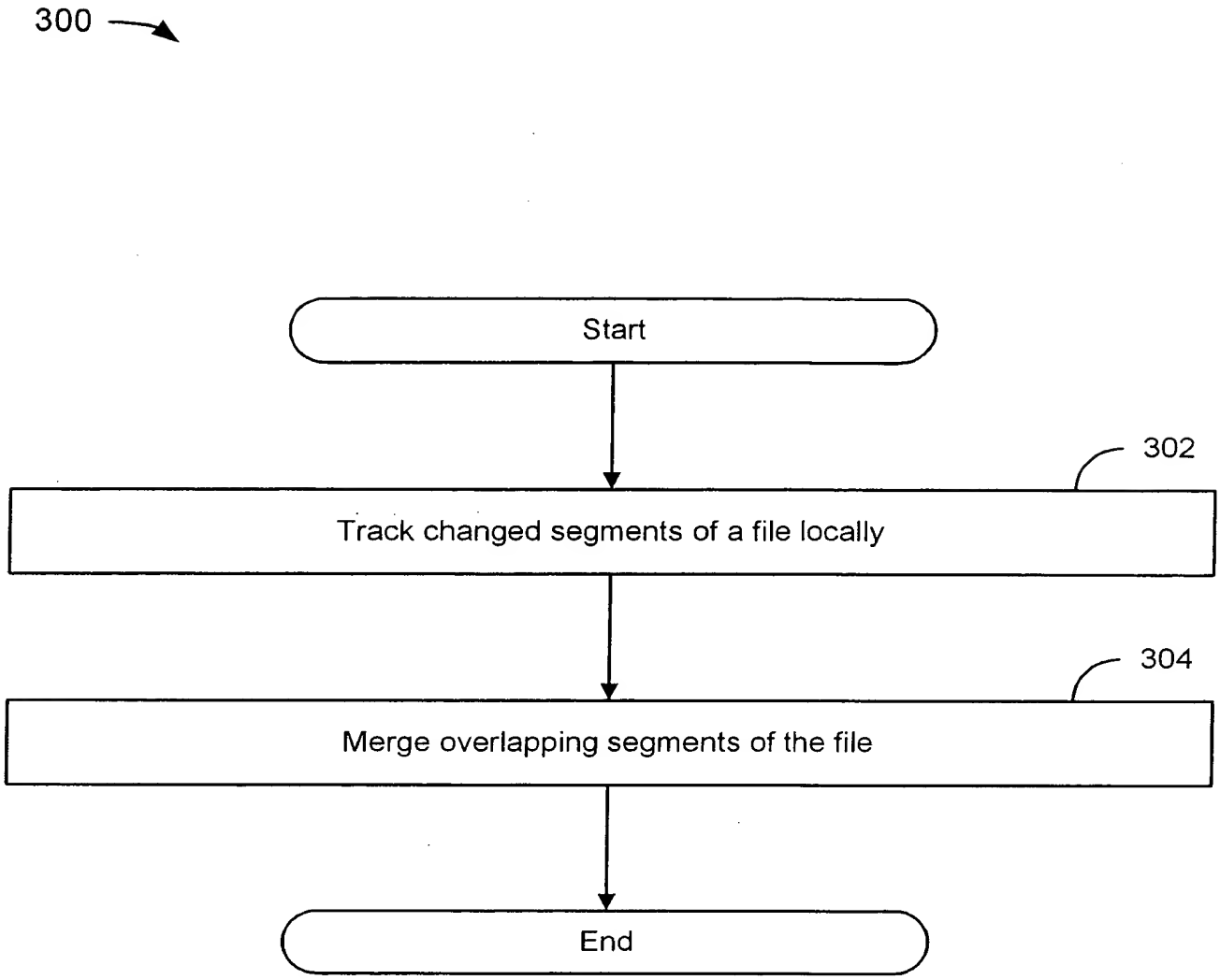
300
Start
302
Track changed segments of a file locally
304
Merge overlapping segments of the file
End

FIG. 3

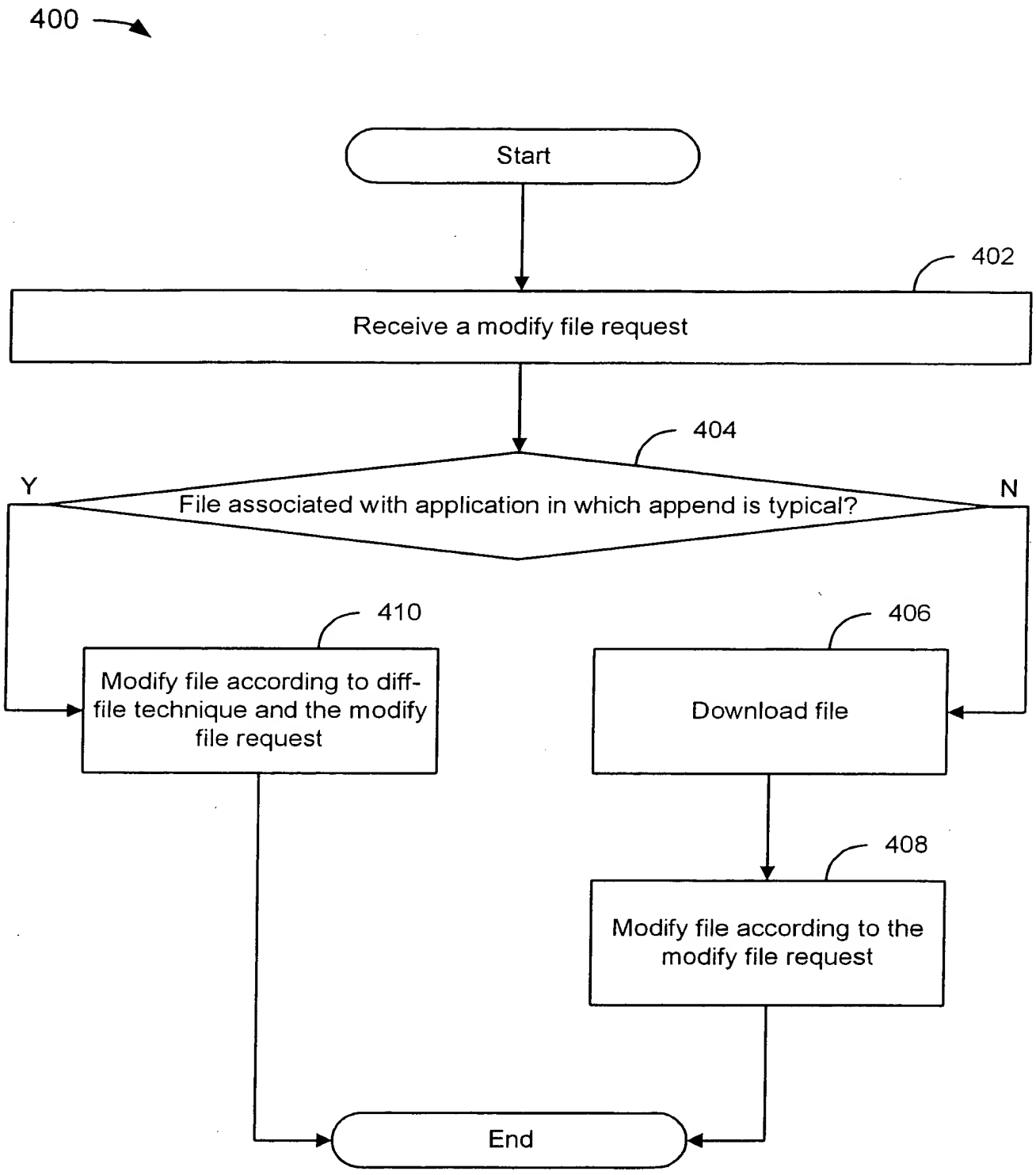
400
Start
402
Receive a modify file request
404
File associated with application in which append is typical?
Y
N
410
Modify file according to diff-file technique and the modify file request
406
Download file
408
Modify file according to the modify file request
End

FIG. 4

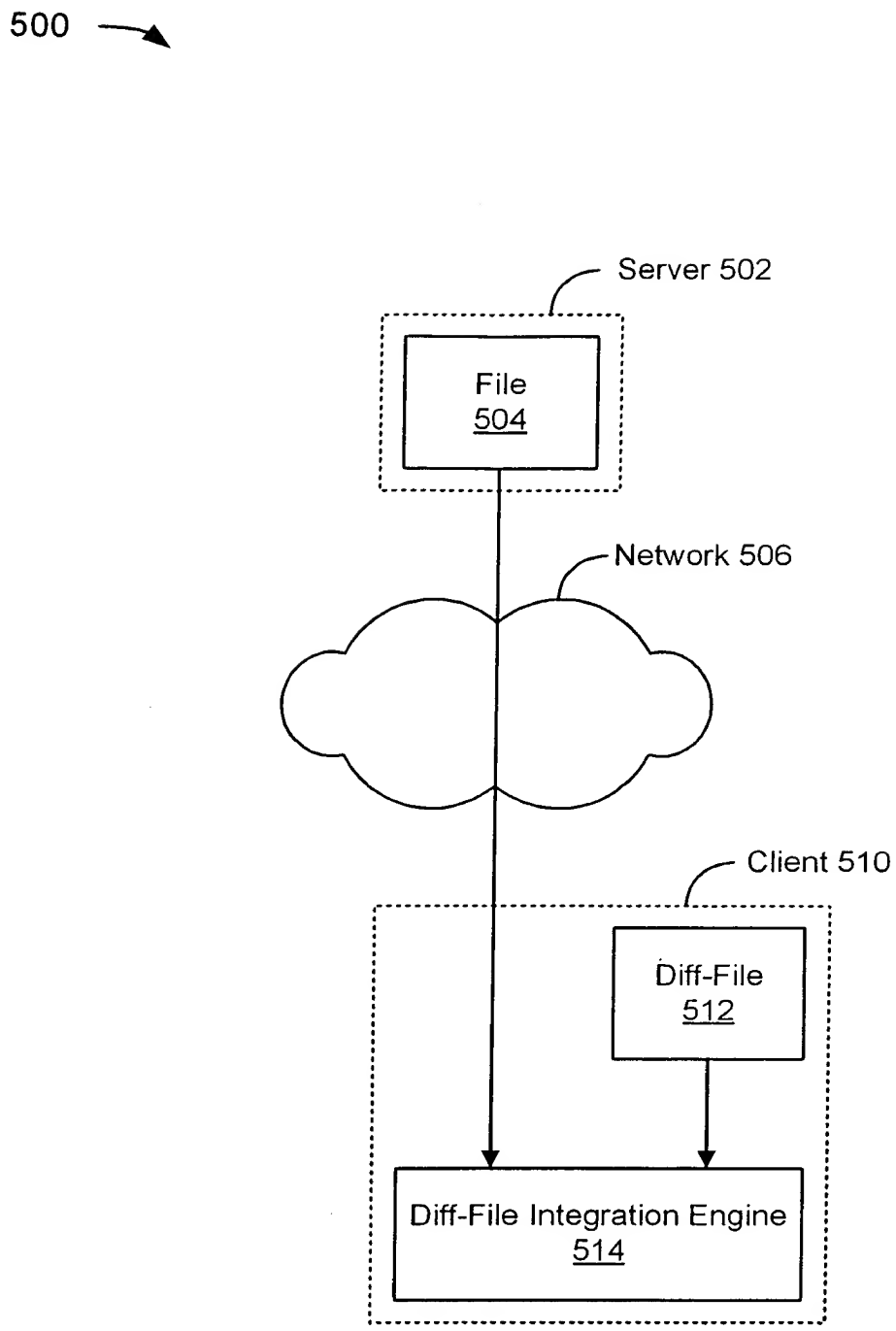
500
Server 502
File
504
Network 506
Client 510
Diff-File
512
Diff-File Integration Engine
514

FIG. 5

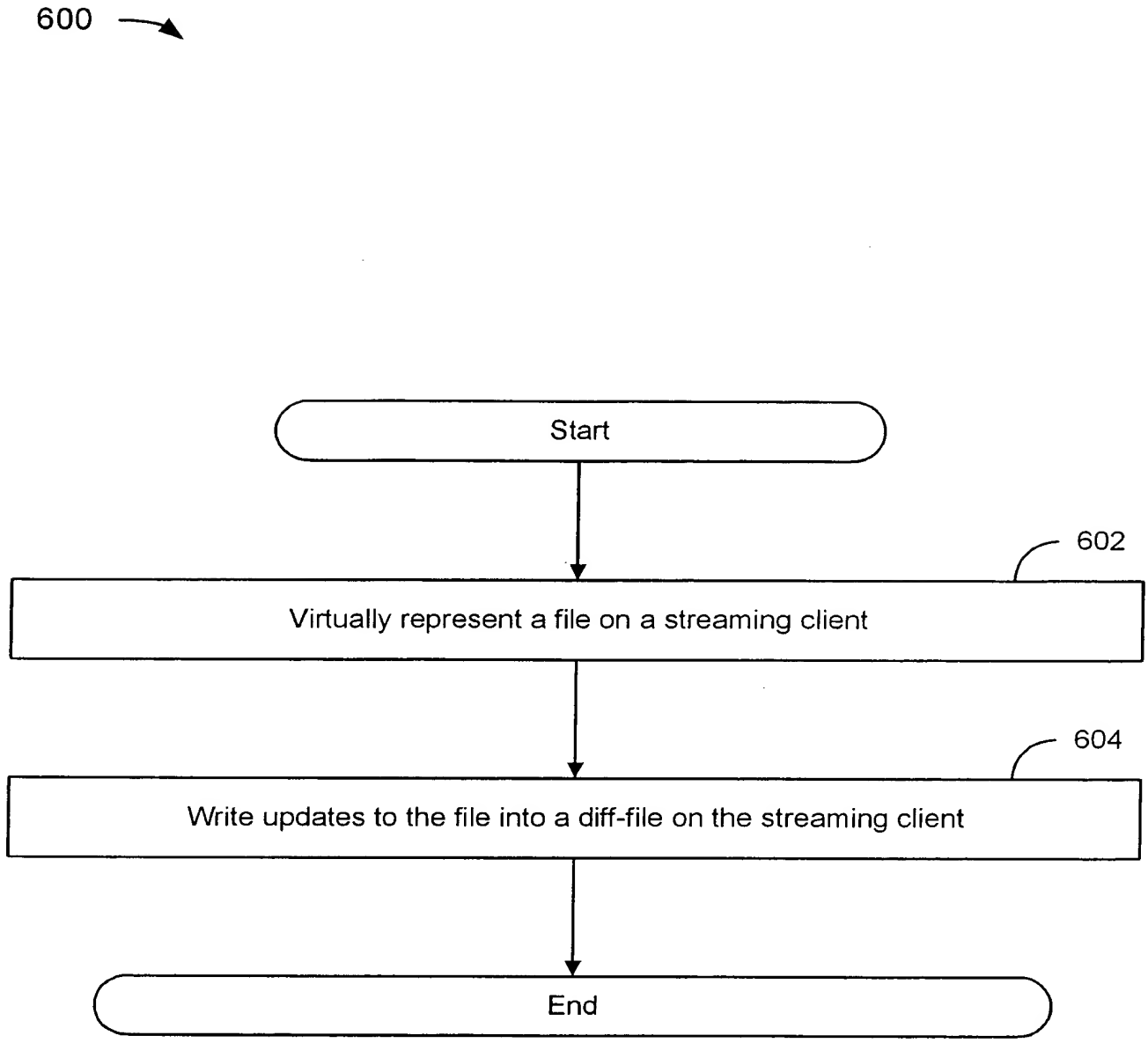
600
Start
602
Virtually represent a file on a streaming client
604
Write updates to the file into a diff-file on the streaming client
End

FIG. 6

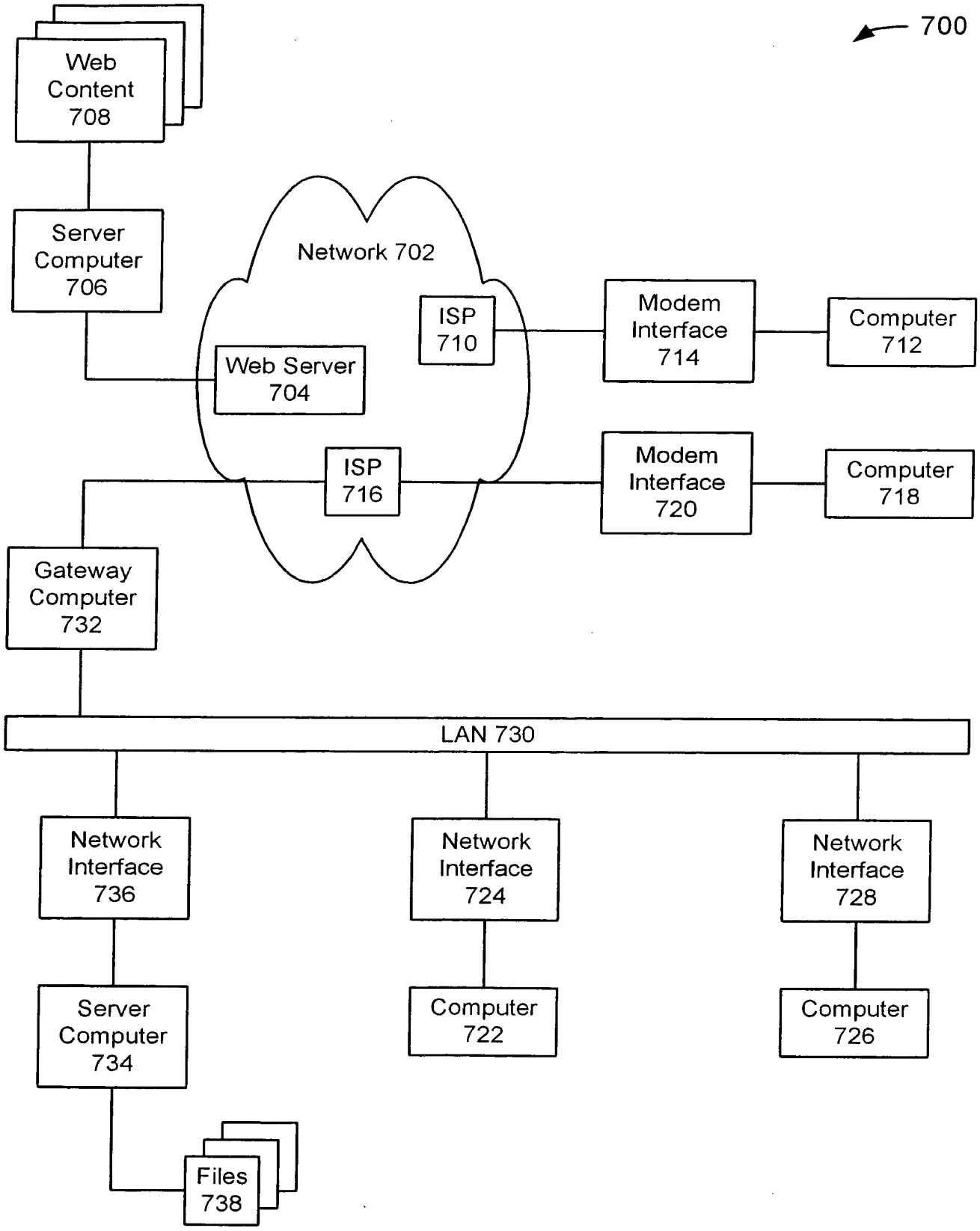
700
Web Content 708
Server Computer 706
Network 702
Web Server 704
ISP 710
Modem Interface 714
Computer 712
ISP 716
Modem Interface 720
Computer 718
Gateway Computer 732
LAN 730
Network Interface 736
Network Interface 724
Network Interface 728
Server Computer 734
Computer 722
Computer 726
Files 738

FIG. 7

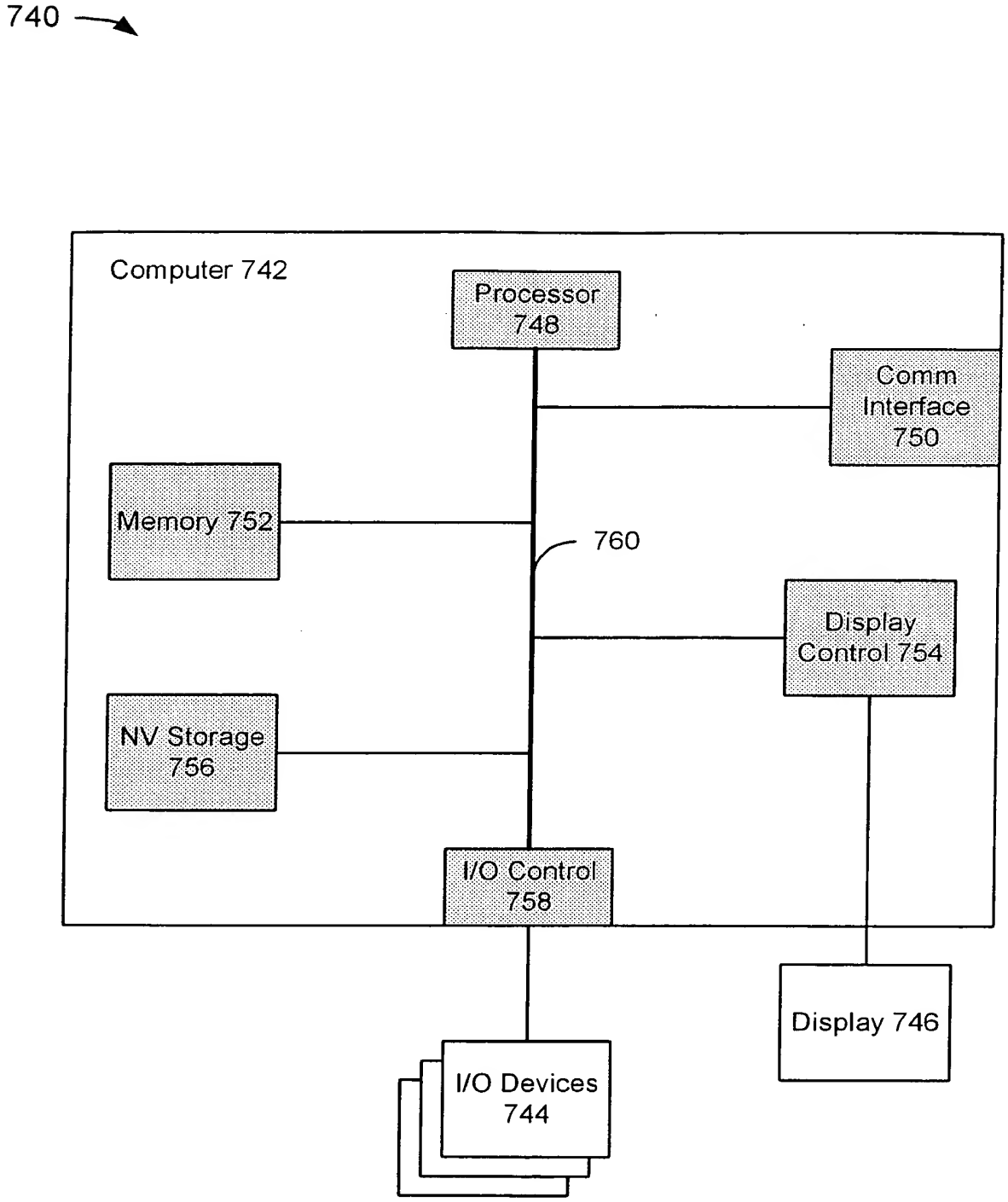
740
Computer 742
Processor 748
Comm Interface 750
Memory 752
760
Display Control 754
NV Storage 756
I/O Control 758
Display 746
I/O Devices 744

FIG. 8